Модуль 2.1. Простейшие конструкции языка С

Темы модуля

- История развития языка программирования C
- Переменные и константы
- Базовые типы данных языка C и их модификация
- Простые операторы: оператор присваивания, арифметические и логические операторы, операторы сравнения
- Выражения: порядок вычислений, преобразование типов в выражениях
- Структура программы на языке C

Планируемые результаты обучения

После изучения данного модуля Вы должны уметь:

на уровне знаний:

- воспроизводить алфавит и лексику языка
- воспроизводить типы данных языка программирования
- воспроизводить правила записи выражений и операций
- воспроизводить синтаксис простых операторов
- описывать структуру программы на языке С

на уровне понимания:

- объяснять применение типов данных

на уровне применения:

- использовать по назначению базовые типы данных языка программирования при объявлении переменных
- записывать в соответствии с правилами языка программирования выражения и операции
- записывать действия алгоритма на языке С в соответствии с синтаксическими правилами записи операторов

на уровне анализа:

- анализировать разработанную программу с целью выявления логических ошибок;

на уровне синтеза:

- использовать математические методы и вычислительные алгоритмы для решения практических задач
- проектировать структуру программы
- организовать работу в группе при совместном решении задачи
- проектировать тестирование программы
- защищать выполненную самостоятельную работу
- принимать верное решение при коллективном решении задачи

Происхождение языка программирования С

- Язык С был изобретен и реализован Деннисом Ритчи (Dennis Ritchie) для компьютера DEC PDP-11 в операционной системе Unix
- Этот язык был разработан на основе "более старого" языка BCPL, созданного в свое время Мартином Ричардсом (Martin Richards)
- BCPL оказал определенное влияние на язык B, разработанный Кеном Томпсоном (Ken Thompson)
- В свою очередь развитие языка B привело к созданию в 1970 году языка С

язык BCPL → язык B → язык C → язык C++

История развития языка программирования С

Год	Событие
1978	Вышла книга Брайана Кернигана(Brian Kernighan) и Денниса Ритчи «The C Programming Language», содержащая описание языка С для операционной системой Unix. Многие года эта версия языка была фактическим стандартом С
Лето 1983	Образован комитет Национального института стандартизации США (American National Standards Institute - ANSI), целью которого была разработка стандарта языка С
Декабрь 1989	Стандарт ANSI был окончательно одобрен
Начало 1990	Стандарт ANSI впервые опубликован Стандарт был также принят организацией ISO (International Standards Organization - Международная организация по стандартизации), поэтому он называется *ANSI/ISO Standard C* Версию С, определенную стандартом 1989 года, обычно называют *C89*
1989-1998	Создание и развитие языка С++, базовым подмножеством которого является стандарт С89
1999	Появился новый стандарт языка С, называемый С99

Свойства языка

- Язык С – язык среднего уровня – объединяет лучшие свойства языков высокого уровня , возможности управления и гибкость язык ассемблера

Языки высокого уровня	**Ada** **Modula-2** **Pascal** **COBOL** **FORTRAN** **Basic**
Языки среднего уровня	Java C++ C FORTH
Языки низкого уровня	Макроассемблер Ассемблер

Свойства языка

- Язык С позволяет манипулировать битами, байтами и адресами, то есть теми базовыми элементами данных, с которыми работает компьютер. Несмотря на это программа, написанная на С, обладает высокой *машинонезависимостью.* Делает язык С очень удобным для системного программирования
- Язык С имеет несколько встроенных типов данных, однако он *не является строго типизированным языком*, как Pascal или Ada. В языке С допускаются почти все преобразования типов
- В отличие от большинства языков высокого уровня, в С почти отсутствует контроль ошибок в процессе выполнения программы. Например, не проверяется нарушение границ массивов. Ответственность за подобные ошибки полностью возлагается на программиста
- С не требует строгой совместимости параметров и аргументов функций. В языках программирования высокого уровня обычно необходимо, чтобы тип аргумента более или менее соответствовал типу параметра. Для С это не характерно, здесь аргумент может иметь почти любой тип, если его можно разумно преобразовать в тип параметра. Более того, компилятор С автоматически осуществляет все виды необходимых преобразований
- Язык имеет малое количество ключевых слов, составляющих его команды. В С89 определено 32 ключевых слова, причем в С99 добавлено только 5 слов. Для сравнения, например, в большинстве версий языка Basic их количество превышает 100!

С – хорошо структурированный язык

- Отличительной особенностью структурированного языка является *отдельное размещение* кода программы и данных. Это позволяет выделять и скрывать от остальной части программы данные и инструкции, необходимые для решения конкретной задачи. Этого можно достичь с помощью подпрограмм с локальными переменными. Используя, локальные переменные, можно создавать подпрограммы, не порождающие побочных эффектов в других модулях
- Структурированные языки допускают использование вложенных циклов
- Использование оператора goto либо запрещено, либо нежелательно
- Структурированные языки позволяют размещать несколько инструкций программы в одной строке
- Структурированные языки считаются более современными

Основные структурные элементы языка С

- *Функции* - это строительные блоки, из которых создается программа. Функции позволяют разбить программу на модули, решающие различные задачи. Написав правильно функцию, можно быть уверенным в ее надежной работе в различных ситуациях без побочных эффектов в других частях программы. При работе над большим проектом, когда особенно важно, чтобы одна часть кода ни в коем случае не могла непредвиденно подействовать на другую часть, умение создать отдельную функцию приобретает для программиста исключительное значение.
- *Программный блок* — это логически связанная группа операторов программы, которую можно рассматривать как отдельную программную единицу. В языке С блок представляет собой последовательность операторов программы, заключенную в фигурные скобки { }. Использование программных блоков позволяет сделать программу понятной, элегантной и эффективной. Более того, программные блоки помогают лучше формализовать задачу и более точно запрограммировать алгоритм ее решения.

Синтаксис и семантика языка программирования

Содержательно язык программирования - это средство общения между человеком (программистом) и компьютером (исполнителем). Рассматривая любую знаковую систему (в том числе и язык программирования), обычно выделяют :

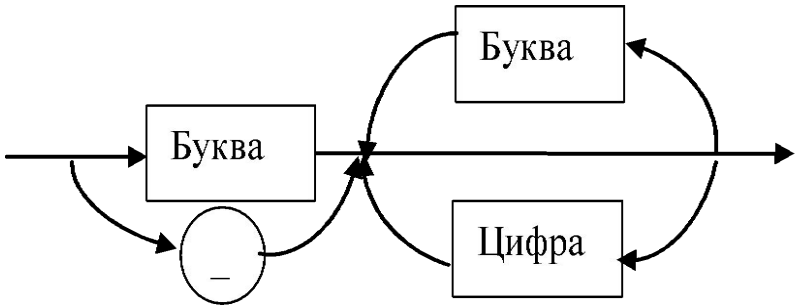
- *синтаксис* - правила построения предложений в этой системе
- *семантику* - правила истолкования предложений тем, кому они адресованы
- *прагматику*, сопоставляющую предложения желаниям того, от кого они исходят

Выражения

- Выражения – самые важные элементы языка С
- Выражения состоят из атомарных элементов: данных и операторов
- Данные представляют собой переменные или константы
- Выражения могу быть арифметические и логические
- В состав арифметических выражений могут входить переменные числового типа и числа; над переменными и числами могут производиться различные арифметические операции, а также математические операции, выраженные с функциями. Вычисление арифметических выражений производится в соответствии с общеизвестным порядком выполнения арифметических операций, который может изменяться с помощью скобок
- В состав логических выражений могут входить логические переменные, а также числа, числовые выражения, которые сравниваются между собой с использованием операций сравнения (>, <, =). Логическое выражение может принимать два значения: «истина» или «ложь». Над элементами логических выражений могут производиться логические операции (И, ИЛИ, НЕ)

Идентификаторы

- В языке С имена переменных, функций, меток и других объектов, определенных пользователями называются идентификаторами
- Длина идентификатора (количество символов, из которых состоит идентификатор) является натуральным числом, обычно идентификатор представляет собой последовательность из одного или нескольких символов. Первый символ должен быть буквой или символом подчеркивания, последующие символы должны быть буквами, цифрами или символами подчеркивания
- На рисунке приведена синтаксическая диаграмма <Идентификатор>



Правила записи идентификаторов

- В языке С длина идентификатора может быть любой, однако не все его символы будут значащими. Рассмотрим это на примере внешних и внутренних идентификаторов:
  - Внешние идентификаторы участвуют в процессе редактирования внешних связей и обозначают имена функций и глобальных переменных, которые используются совместно в различных исходных файлах
  - Внутренние идентификаторы обозначают имена локальных переменных

  В стандарте С89 значащими являются как минимум первые 6 символов внешнего имени и первые 31 символ внутреннего имени

  В стандарте С99 значащими являются для внешнего идентификатора первые 31 символ, а для внутреннего — первые 63 символа

  В С++ значащими являются как минимум 1024 символа любого идентификатора

  Эти отличия необходимо учитывать при конвертировании программ, написанных на языках С89, С99 или просто С, в программы на С++
- Верхние и нижние регистры символов рассматриваются как различные. Следовательно, count, Count и COUNT — это три разных идентификатора.
- Идентификатор не может совпадать с ключевым словом С или с именем библиотечной функции

Примеры правильных и неправильных записей идентификаторов

- ***Правильные***

Count

count1

test23

high_balance

- ***Неправильные***

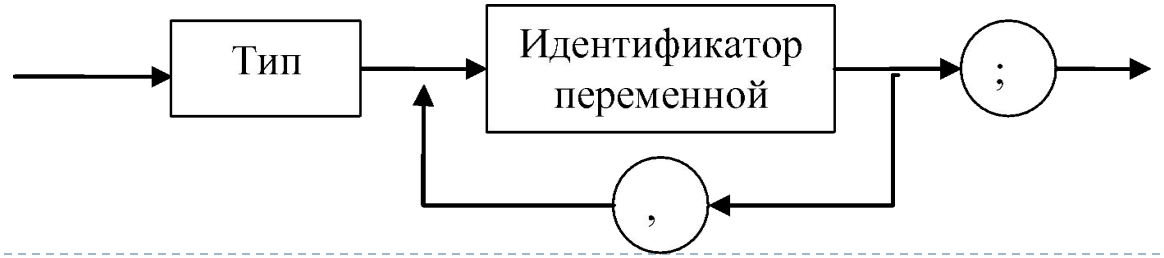
1count

hi!here

high...balance

Переменные

- Переменная - представляет собой имя ячейки памяти, которую можно использовать для хранения модифицируемого значения
- Все переменные должны быть объявлены до своего использования
- Общая форма *объявления* имеет такой вид:



Пять основных типов данных

Тип данных – фундаментальное понятие теории программирования. Тип данных определяет *множество значений*, *набор операций*, которые можно применять к таким значениям, и, возможно, *способ реализации хранения значений и выполнения операций*. Любые данные, которыми оперируют программы, относятся к определённым типам.

Стандарт C89 определяет пять ***базовых типов данных***:

- char – символьные данные
- int – целые
- float – число с плавающей точкой
- double – число с плавающей точкой двойной точности
- void – переменная, не имеющая значения

Тип void служит для объявления функции, не возвращающей значения, или для создания универсального указателя

На основе этих типов формируются другие типы данных

Размеры и диапазоны значений типов данных

Размер (объем занимаемой памяти) и диапазон значений этих типов данных для разных процессоров и компиляторов могут быть разными:

- объект типа char всегда занимает 1 байт
- размер объекта int обычно совпадает с размером слова в конкретной среде программирования. В большинстве случаев в 16-разрядной среде (DOS или Windows 4.1) int занимает 16 битов, а в 32-разрядной (Windows 95/98/NT/2000) — 32 бита
- Стандарт С обусловливает только минимальный диапазон значений каждого типа данных, но не размер в байтах
- Конкретный формат числа с плавающей точкой зависит от его реализации в трансляторе. Переменные типа char обычно используются для обозначения набора символов стандарта ASCII, символы, не входящие в этот набор, разными компиляторами обрабатываются по-разному
- Диапазон значений типов float и double зависит от формата представления чисел с плавающей точкой. Стандарт С определяет для чисел с плавающей точкой минимальный диапазон значений от 1E-37 до 1E+37

Модификаторы типов

- Базовые типы данных (кроме void) могут иметь различные модификаторы, предшествующие им в тексте программы
- Модификатор типа так изменяет значение базового типа, чтобы он более точно соответствовал своему назначению в программе. Полный список спецификаторов типов:

  signed

  unsigned

  long

  short
- Базовый тип int может быть модифицирован каждым из этих спецификаторов
- Тип char модифицируется с помощью unsigned и signed, double — с помощью long
- В таблице приведены все допустимые комбинации типов данных с их минимальным диапазоном значений и типичным размером. Обратите внимание, в таблице приведены *минимально возможные*, а не типичные диапазоны значений

Типы данных, определенные стандартом ANSI/ISO Standard C

Тип	Типичный размер в битах	Минимально допустимый диапозон значений
char	8	от -127 до 127
unsigned char	8	от 0 до 255
signed char	8	от -127 до 127
int	16 или 32	от -32767 до 32767
unsigned int	16 или 32	от 0 до 65535
signed int	16 или 32	то же, что int
short int	16	от -32767 до 32767
unsigned short int	16	от 0 до 65535
signed short int	16	то же, что short int
long int	32	от -2 147 483 647 до 2 147 483 647
long long int	64	от $-(2^{63}-1)$ до $(2^{63}-1)$, добавлен стандартом C99
signed long int	32	то же, что long int
unsigned long int	32	от 0 до 4 294 967 295
unsigned long long int	64	от 0 до $(2^{64}-1)$, добавлен в C99
float	32	от 1E-37 до 1E+37, с точностью не менее 6 значащих десятичных цифр
double	64	от 1E-37 до 1E+37, с точностью не менее 10 значащих десятичных цифр
long double	80	от 1E-37 до 1E+37, с точностью не менее 10 значащих десятичных цифр

Объявление переменных

- Ниже приведены примеры объявлений переменных:
  int i,j,l;
  short int si;
  unsigned int ui;
  double balance, profit, loss;
- Необходимо помнить, что в С имя переменной никогда не определяет ее тип
- Объявление переменных может быть расположено в трех местах: внутри функции, в определении параметров функции и вне всех функций
- Это - места объявлений соответственно локальных переменных, формальных параметров функций и глобальных переменных

Локальные переменные

- Переменные, объявленные внутри функций, называются локальными переменными
- Локальную переменную можно использовать только внутри блока, в котором она объявлена, то есть она невидима за пределами своего блока
- Локальные переменные существуют только во время выполнения программного блока, в котором они объявлены, то есть создаются они при входе в блок, а разрушаются — при выходе из него. Более того, переменная, объявленная в одном блоке, не имеет никакого отношения к переменной с тем же именем, объявленной в другом блоке

Константы

- Константы – фиксированные значения, которые программа не может изменить
- Способ представления константы зависит от ее типа
- Символьные константы заключаются в одинарные кавычки - ‘a’, ‘+’
- Целочисленные константы – числа, не имеющие дробной части. Например, 10 и -100 — это целые константы. Константы в плавающем формате записываются как числа с десятичной точкой, например, 11.123. Допускается также экспоненциальное представление чисел (в виде мантиссы и порядка): 111.23e— 1.
- Строковые константы - это последовательность символов, заключенных в двойные кавычки. Например, "тест" - это строка
- Шестнадцатеричные и восьмеричные константы: префикс 0x ставится перед шестнадцатеричным числом и 0 – перед восьмеричным

```
int hex=0x80 /*128 в десятичной системе*/
int oct=012 /*10 вдесятичной системе */
```

- Специальные символьные константы - некоторые символы, например, символ возврата каретки, требуют специального представления. Иногда их называют ESC-последовательностями, управляющими последовательностями и символами с обратным слэшем. Управляющие последовательности можно использовать вместо ASCII-кодов для обеспечения лучшей переносимости программы

Строковые константы

Код	*Назначение*
\b	Удаление предыдущего символа
\f	Подача бумаги
\n	Новая строка
\r	Возврат каретки
\t	Горизонтальная табуляция
\"	Двойные кавычки
\'	Одинарная кавычка
\\	Обратный слэш
\v	Вертикальная табуляция
\a	Сигнал
\?	Знак вопроса
\N	Восьмеричная константа (N - восьмеричное представление)
\xN	Шестнадцатеричная константа (N - шестнадцатеричное представление)

Операции, операторы, операнды

- Язык С содержит большое количество встроенных операций - специальных способов записи действий
- Оператор - наименьшая автономная часть языка программирования (команда) , которую можно выполнить отдельно
- Операнд - аргумент оператора или данные, которые обрабатываются оператором

  В зависимости от положения операнд относительно операции различают префиксные (напр., sinx (x — операнд)), инфиксные (например, a + b (a, b — операнды)) и постфиксные (например, x3 (x — операнд)) операции

  В зависимости от числа операнд различают одноместные (унарные, или монадические) операции (например, -a); двуместные (бинарные, или диадические) операции (a + b); многоместные (или полиадические) операции
- В языке С существует четыре основных класса операторов:
  - арифметические
  - логические
  - поразрядные
  - операторы сравнения

Кроме них, есть также некоторые специальные операторы

Оператор присваивания

Оператор присваивания может присутствовать в любом выражении языка С

Общая форма оператора присваивания:

имя_переменной = выражение;

Выражение может быть просто константой или сколь угодно сложным выражением

Адресатом (получателем), т.е. левой частью оператора присваивания должен быть объект, способный получить значение, например, переменная.

Множественное присваивание

В одном операторе присваивания можно присвоить одно и то же значение многим переменным. Для этого используется оператор *множественного присваивания*, например:

x = y = z = 0;

Составное присваивание

Составное присваивание — это разновидность оператора присваивания, в которой запись сокращается и становится более удобной в написании. Например, оператор

```
x = x+10;
```

можно записать как

```
x += 10;
```

Оператор "+=" сообщает компилятору, что к переменной x нужно прибавить 10.

"Составные" операторы присваивания существуют для всех бинарных операций

Любой оператор вида

```
переменная = переменная оператор выражение;
```

можно записать как

```
переменная оператор = выражение;
```

Преобразование типов в операторе присваивания

- Правило преобразования типов для оператора присваивания: значение правой части преобразовывается к типу левой части

```
int x;
char ch;
float f;
void func(void)
{
ch = x;  /* 1-я строка */
x = f;    /* 2-я строка */
f = ch;  /* 3-я строка */
f = x;    /* 4-я строка */
}
```

- В 1-й строке этого примера старшие двоичные разряды целой переменной x отбрасываются, а в ch заносятся младшие 8 бит. Если значение x лежит в интервале от 0 до 255, то ch и x будут идентичны и потери информации не произойдет. В противном случае в ch будут занесены только младшие разряды переменной x.
- Во 2-й строке в x будет записана целая часть числа f.
- В 3-й строке произойдет преобразование целого 8-разрядного числа, хранящегося в ch, в число в плавающем формате.
- В 4-й строке произойдет то же самое, только с 16-разрядным целым.

Арифметические операции

Оператор	Операция
-	Вычитание, так же унарный минус
+	Сложение
*	Умножение
/	Деление
%	Остаток от деления
--	Декремент, или уменьшение
++	Инкремент, или увеличение

Семантика операторов

Оператор деления по модулю % возвращает остаток от целочисленного деления

! Этот оператор нельзя применять к типам данных с плавающей точкой

Операции увеличения (инкремента) и уменьшения (декремента)

Оператор инкремент ++ увеличивает значение операнда на 1, а декремент -- уменьшает на 1.

Оператор

```
x = x+1;
```

можно записать как

```
++x;
```

Аналогично оператор

```
x = x-1;
```

равносилен оператору

```
x--;
```

Префиксная и постфиксная формы записи инкремента и декремента

Как инкремент, так и декремент могут предшествовать операнду (префиксная форма) или следовать за ним (постфиксная форма).

Например

```
x = x+1;
```

можно записать как в виде

```
++x;
```

так и в виде

```
x++;
```

Префиксная и постфиксная формы отличаются при использовании их в выражениях:

- Если оператор инкремента или декремента предшествует операнду, то сама операция выполняется до использования результата в выражении
- Если же оператор следует за операндом, то в выражении значение операнда используется до выполнения операции инкремента или декремента. То есть для выражения эта операция как бы не существует, она выполняется только для операнда.

Например, `x = 10; y = ++x;`

присваивает y значение 11.

Однако если написать `x = 10; y = x++;`

то переменной y будет присвоено значение 10.

В обоих случаях x присвоено значение 11, разница только в том, когда именно это случилось, до или после присваивания значения переменной y

Приоритет выполнения арифметических операторов

Наивысший

++ --

-(унарный минус)

* / %

Наинизший

+ -

Операции с одинаковым приоритетом выполняются слева направо. Используя круглые скобки, можно изменить порядок вычислений. В языке С круглые скобки интерпретируются компилятором так же, как и в любом другом языке программирования: они как бы придают операции (или последовательности операций) наивысший приоритет

Операции сравнения и логические операции

- Операции сравнения — это операции, в которых значения двух переменных сравниваются друг с другом.
- Логические же операции реализуют средствами языка С операции формальной логики.
- Между логическими операциями и операциями сравнения существует тесная связь: результаты операций *сравнения* часто являются операндами *логических* операций.
- В операциях сравнения и логических операциях в качестве операндов и результатов операций используются значения ИСТИНА (true) и ЛОЖЬ (false).
- В языке С значение ИСТИНА представляется любым числом, отличным от нуля. Значение ЛОЖЬ представляется нулем. Результатом операции сравнения или логической операции являются ИСТИНА (true, 1) или ЛОЖЬ (false, 0).
- Как операции сравнения, так и логические операции имеют низший приоритет по сравнению с арифметическими

Операторы сравнения и логические операторы

Оператор	Операция
>	Больше чем
>=	Больше или равно
<	Меньше чем
<=	Меньше или равно
==	Равно
!=	Не равно

Оператор	Операция
&&	И
||	ИЛИ
!	НЕ, отрицание

Таблица истинности логических операций

p	*q*	*p && q*	*p || q*	*!p*
0	0	0	0	1
0	1	0	1	1
1	1	1	1	0
1	0	0	1	0

Программа на языке С

Программа – последовательность инструкций на языке программирования

Любая программа на С состоит из одной или нескольких функций

Обязательно должна быть определена единственная главная функция main(), именно с нее всегда начинается выполнение программы

Ключевые слова языка С стандарта С89

- Они являются ключевым словами и языка С++

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Структура программы на языке С

```
Директивы препроцессора
Объявление глобальных переменных
тип_возвращаемого_значения main(список параметров)
{
  последовательность операторов
}
тип_возвращаемого_значения f1(список параметров)
{
  последовательность операторов
}
тип_возвращаемого_значения f2(список п
{
  последовательность операторов
}
 . . .
тип_возвращаемого_значения fN(список п
{
  последовательность операторов
}
```

- Директивы препроцессора – инструкции регламентирующие работу компилятора. Их применение расширяет возможности программ
- Все директивы начинаются со знака #
- Глобальные переменные – переменные, объявленные вне всех функций
- Тип возвращаемого значения – любой базовый тип языка

Побитовые (поразрядные) операторы

Поразрядные операции

- В отличие от многих других языков программирования в С определен полный набор *поразрядных операций*. Это обусловлено тем, что С был задуман как язык, призванный во многих приложениях заменить ассемблер, который способен оперировать битами данных
- *Поразрядные операции* — это тестирование (проверка), сдвиг или присвоение значений отдельным битам данных
- *Поразрядные операции* осуществляются над ячейками памяти, содержащими данные типа char или int. Данные типа float, double, long double, void или другие более сложные не могут участвовать в поразрядных операциях

Поразрядные операторы

Оператор	*Операция*
&	И
|	ИЛИ
^	исключающее ИЛИ
~	НЕ (отрицание, дополнение к 1)
>>	Сдвиг вправо
<<	Сдвиг влево

Таблицы истинности

- Поразрядные операции И, ИЛИ, НЕ описываются теми же таблицами истинности, что и логические операции
- Поразрядные операции выполняются над отдельными разрядами (битами) операндов
- Операция "исключающее ИЛИ" имеет следующую таблицу истинности:

p	*q*	*p* ^ *q*
0	0	0
1	0	1
1	1	0
0	1	1

Применение поразрядных операторов

- при программировании драйверов устройств, таких как модемы
- при программировании процедур, выполняющих операции над файлами
- при разработке стандартных программ обслуживания принтера

Побитовый оператор И: &

- Операция & может быть использована для *очищения (сбрасывания значения) бита*
- Любой бит одного из операндов, равный 0, обнуляет значение соответствующего бита в результате
- Например, следующая функция читает символ из порта модема и обнуляет бит контроля четности:

```
char get_char_from_modem(void)
{
    char ch;
    ch = read_modem(); /* чтение символа из порта модема */
return(ch & 127);
}
```

Бит контроля четности, находящийся в 8-м разряде байта, обнуляется с помощью операции И. При этом в качестве второго операнда выбирается число, имеющее 1 в разрядах от 1 до 7, и 0 в 8-м разряде. Именно таким числом и является 127, поскольку все биты двоичного представления числа 127, кроме старшего, равны 1. Выражение ch & 127 означает попарное применение операции И ко всем битам, составляющим значение переменной ch, и битам числа 127. В результате все биты, кроме старшего, остаются без изменения, а старший обнуляется:

```
Бит контроля четности  - 8 бит
  1100 0001     переменная ch содержит символ 'A' с битом четности
  0111 1111     двоичное представление числа 127
& ---------     поразрядная операция И
  0100 0001     символ 'A' с обнуленным битом контроля четности
```

Поразрядная операция ИЛИ: |

- Поразрядная операция ИЛИ применяется для установки необходимых битов в 1
- В следующем примере выполняется операция 128 | 3:

```
1000 0000     двоичное представление числа 128
0000 0011     двоичное представление числа 3
| ---------    поразрядная операция ИЛИ
1000 0011     результат
```

Операция исключающего ИЛИ (XOR): ^

- Операция исключающего ИЛИ устанавливает бит результата в 1, если соответствующие биты операндов различны
- В следующем примере выполняется операция 127 ^ 120:

```
0000 0011     двоичное представление числа 127
0111 1000     двоичное представление числа 120
^ ---------    поразрядная операция XOR
0000 0111     результат
```

Необходимо помнить, что результат логической операции всегда равен 0 или 1. В то же время результатом поразрядной операции может быть любое значение, которое, как видно из предыдущих примеров, не обязательно равно 0 или 1.

- Результат логической операции всегда равен 0 или 1
- Результат поразрядной операции может быть любое равен любому целому числу, не обязательно равно 0 или 1.

Поразрядные операторы сдвига >> и <<

- Поразрядные операторы сдвига >> и << смещают все биты переменной вправо или влево на указанное количество разрядов
- Общая форма оператора сдвига вправо:

  переменная >> количество_разрядов
- Общая форма оператора сдвига влево:

  переменная << количество_разрядов
- Как только сдвигаемые биты достигают края, с противоположного конца появляются нули
- Если число типа signed int отрицательно, то при сдвиге вправо левый конец заполняется единицами, так что знак числа сохраняется
- Если биты исчезают на одном краю числа, они не появятся на другом
- Поразрядные операции сдвига очень полезны при декодировании информации, поступающей с внешнего устройства
- Побитовые операторы сдвига можно использовать для быстрого умножения или деления целых чисел: сдвиг на один бит вправо делит число на 2, а на один бит влево — умножает на 2

Умножение и деление операторами сдвига

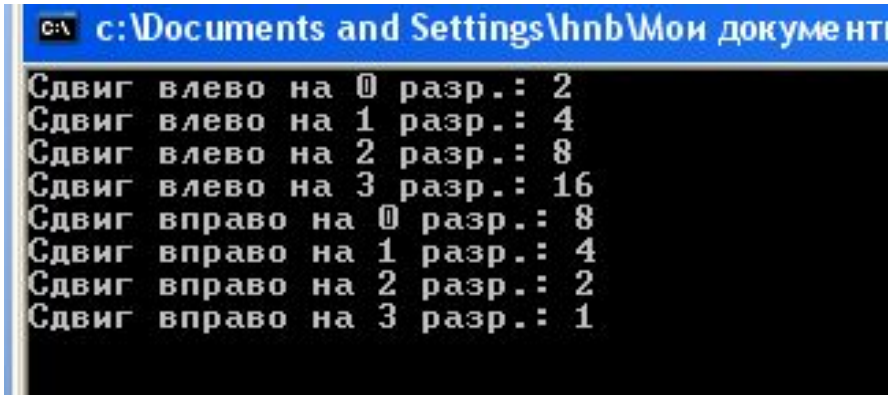
unsigned char x	*x после операции*	*значение x*
x = 7	0000 0111	7
x = x << 1	0000 1110	14
x = x << 3	0111 0000	112
x = x << 2	1100 0000	192
x = x >> 1	0110 0000	96
x = x >> 2	0001 1000	24

Каждый сдвиг влево умножает на 2. Потеря информации произошла после операции x << 2 в результате сдвига за левую границу.
Каждый сдвиг вправо делит на 2. Сдвиг вправо потерянную информацию не восстановил.

Пример применения операторов сдвига.

```
#include <stdio.h>
#include <conio.h>
#include "locale.h"
int main(void)
{
  setlocale(LC_ALL, "rus");
     unsigned int i;
  int j;
  i = 1;
  /* сдвиг влево */
  for(j=0; j<4; j++) {
    i = i << 1;  /* сдвиг i влево на 1 разряд, что
     равносильно умножению на 2 */
    printf("Сдвиг влево на %d разр.: %d\n", j, i);
  }

  /* сдвиг вправо */
  for(j=0; j<4; j++) {
    i = i >> 1;  /* сдвиг i вправо на 1 разряд, что
                    равносильно делению на 2 */
    printf("Сдвиг вправо на %d разр.: %d\n", j, i);
  }
getch();
  return 0;
}
```



Поразрядная операция отрицания ~

- Поразрядная операция отрицания (дополнения до единицы) ~ инвертирует состояние каждого бита операнда.
- То есть, 0 преобразует в 1, а 1 — в 0.
- Поразрядные операции часто используются в шифровальных программах. Проделав с дисковым файлом некоторые поразрядные операции, его можно сделать нечитаемым. Простейший способ сделать это — применить операцию отрицания к каждому биту:

  Исходный байт 0010100

  После 1-го отрицания 1101011

  После 2-го отрицания 0010100

- Обратите внимание, при последовательном применении 2-х отрицаний результатом всегда будет исходное число. Таким образом, 1-е отрицание кодирует состояние байта, а 2-е — декодирует

Пример использования операции отрицания

- В следующем примере оператор отрицания используется в функции шифрования символа:

```
#include <stdio.h>
#include <conio.h>
#include "locale.h"
int main(void)
{
  setlocale(LC_ALL, "rus");
char ch;
scanf(«%c", &ch);
printf(«%c",~ch); /* операция отрицания */
getch();
  return 0;
}
```